Ανάπτυξη ασφαλούς λογισμικού κυβερνοφυσικού συστήματος

ΙΩΝ-ΑΘΑΝΑΣΙΟΣ ΜΕΡΚΟΥΡΗΣ

Επιβλέπων καθηγητής : Δημήτρης Σερπάνος

Introduction:

Safety-Critical Cyber-Physical Systems

- Formal verification of CPS
- Safety & correctness guarantees
- Case studies: Insulin & Agriculture

Problem Motivation

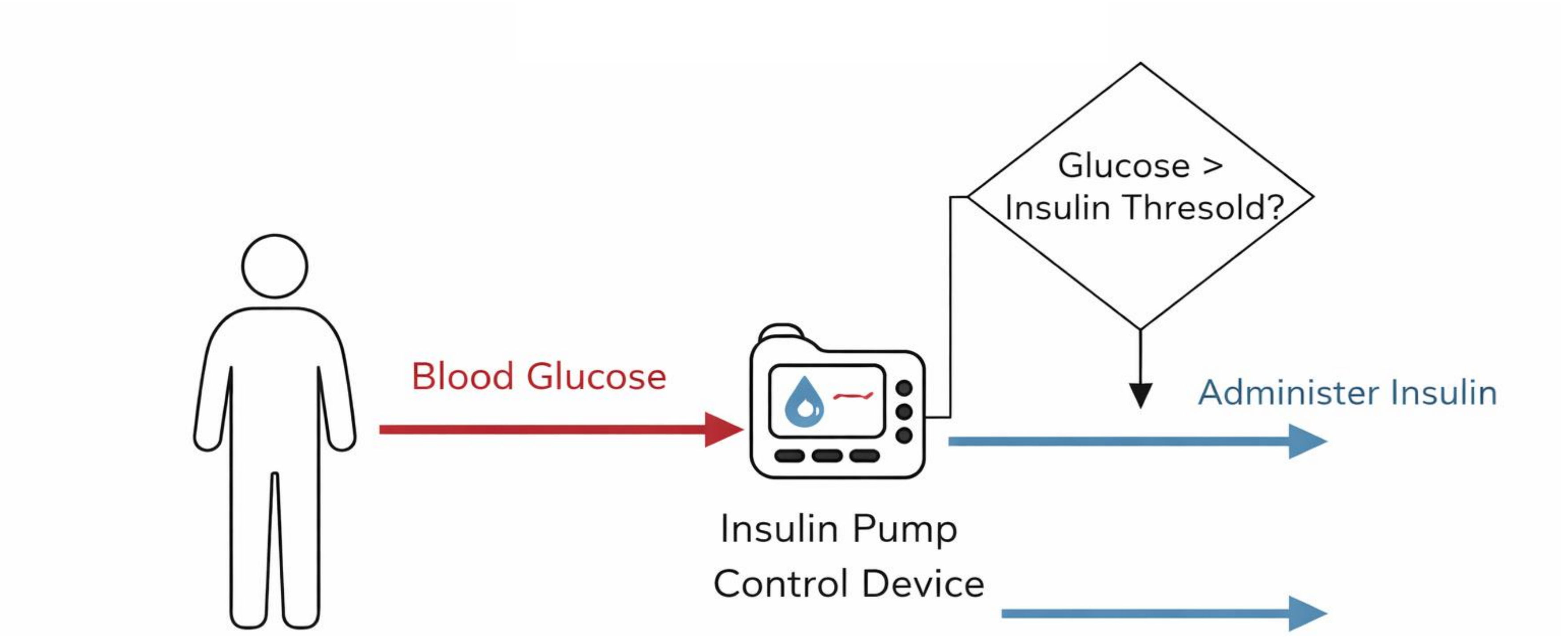
- Controllers interact with physical processes
- Sensors may fail or give noisy data
- Wrong decisions may cause physical or economic harm

Solution: Methodology & Tools

- Formal modeling with F*
- Refinement types
- Pre/Post-conditions and lemmas

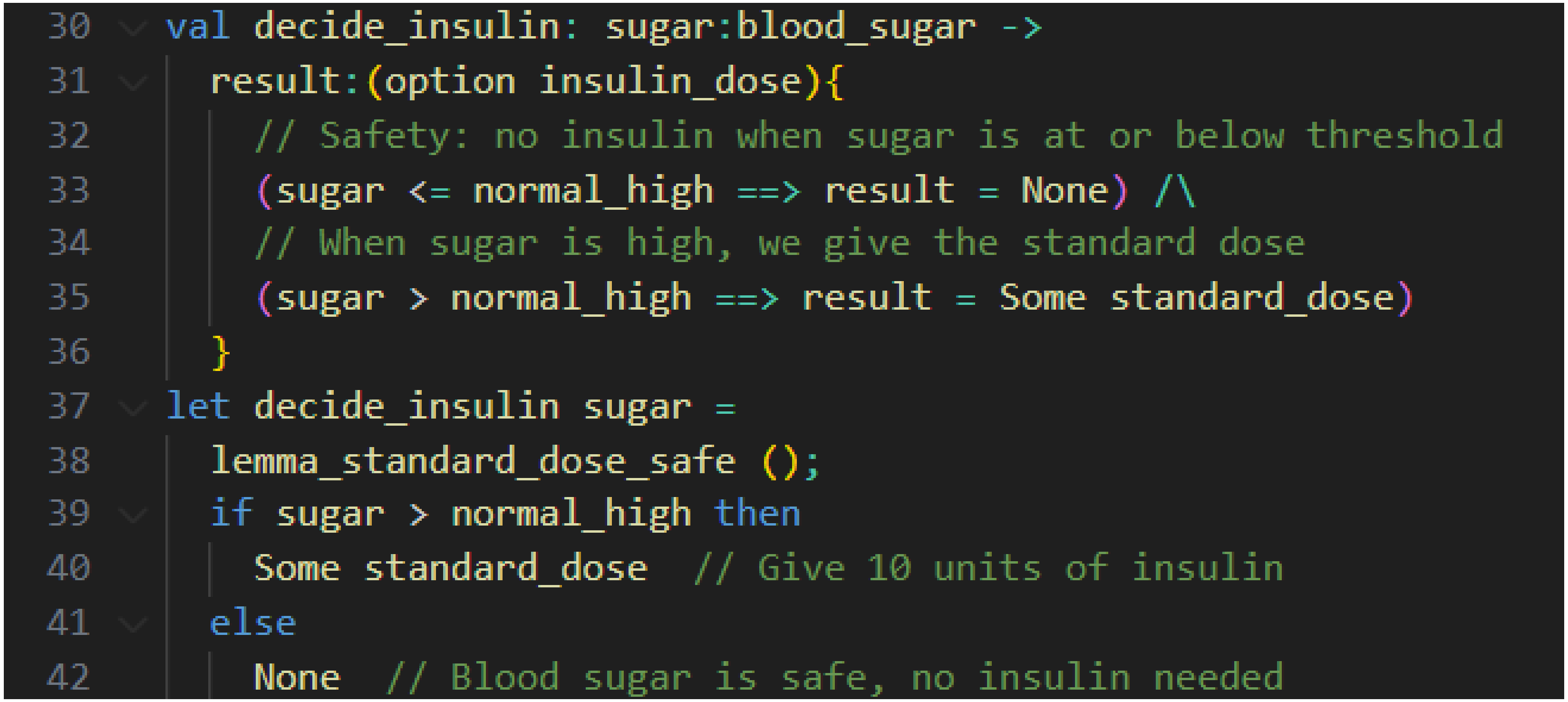
System 1: Insulin Dose System

- Medical safety-critical example
- Decision based on blood sugar level
- Avoid hypoglycemia



Insulin System – Safety Guarantees

- No insulin when sugar ≤ threshold
- Dose always within safe bounds
- Formally proven properties

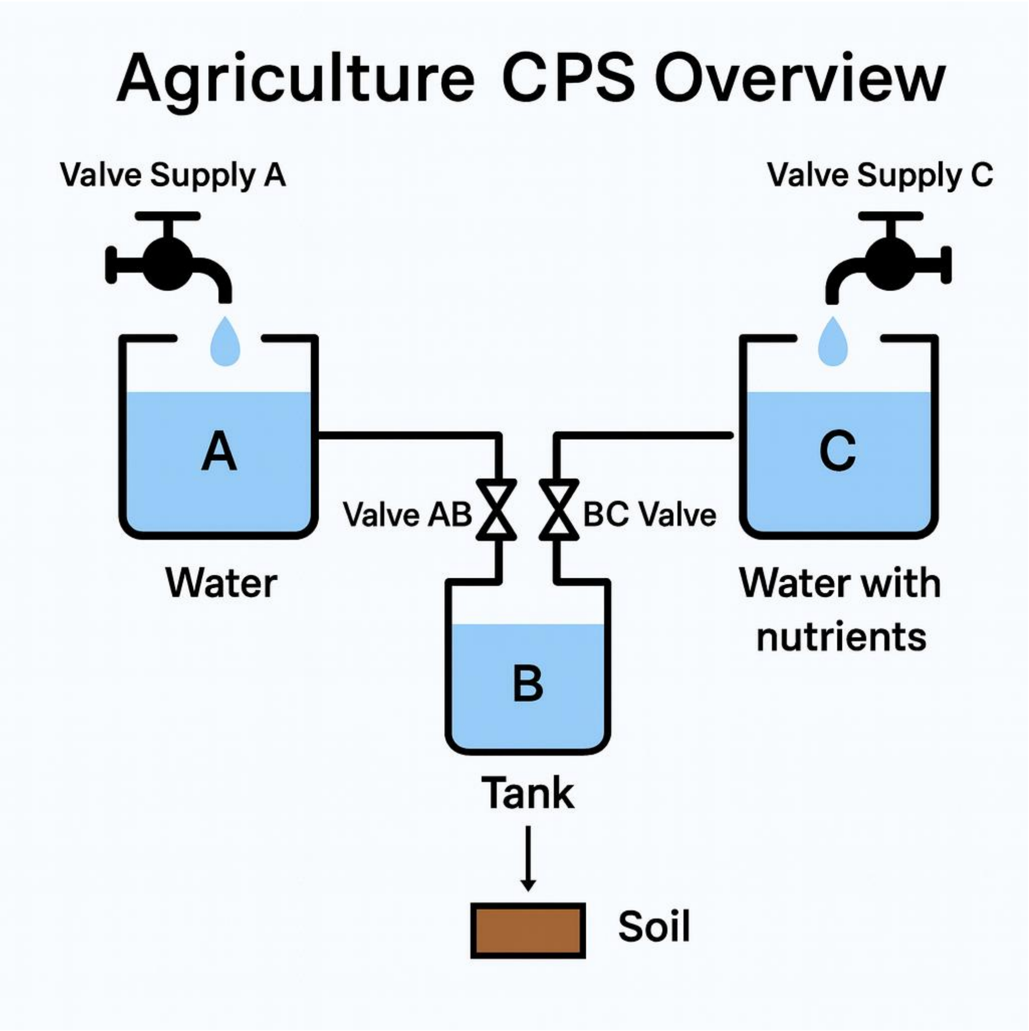


```
val decide_insulin: sugar:blood_sugar ->
  result:(option insulin_dose){
    // Safety: no insulin when sugar is at or below threshold
    (sugar <= normal_high ==> result = None) /\
    // When sugar is high, we give the standard dose
    (sugar > normal_high ==> result = Some standard_dose)
  }
let decide_insulin sugar =
  lemma_standard_dose_safe ();
  if sugar > normal_high then
    Some standard_dose  // Give 10 units of insulin
  else
    None  // Blood sugar is safe, no insulin needed
```

Transition to Agriculture CPS

- From simple to complex CPS
- Multiple components and interactions
- Richer physical dynamics

Agriculture System – Architecture

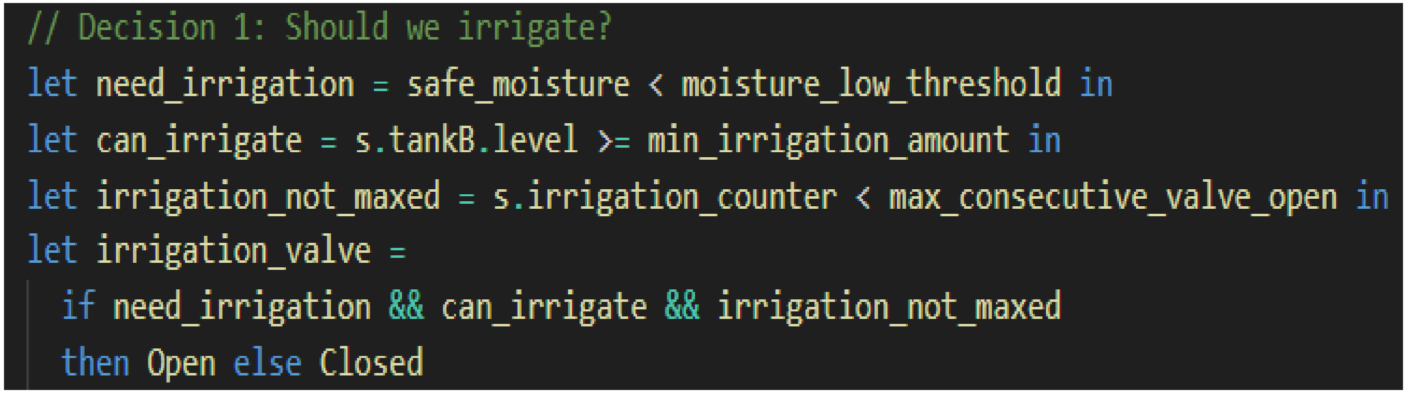


System Components

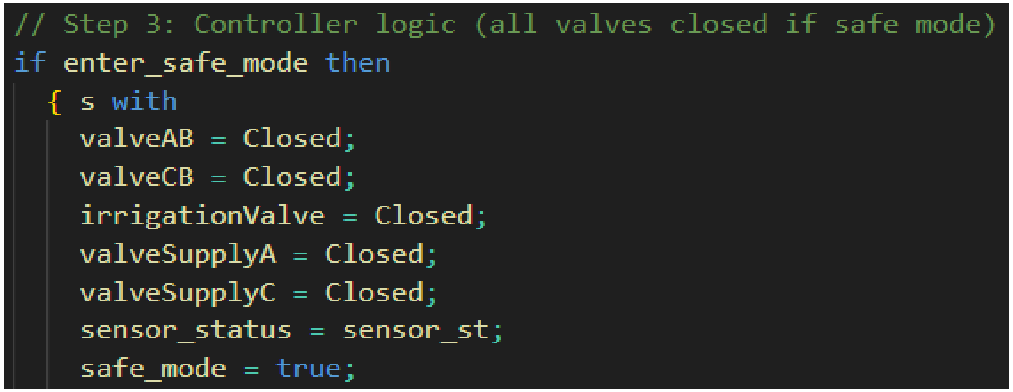
- **Tanks (A, B, C)** – water and nutrient storage
- **Valves** – controlled flows between components
- **Sensors** – soil moisture & environment
- **Controller** – decision logic & safety enforcement

Controller Logic

- Irrigation threshold *(irrigation valve)*
- Target moisture *(nutrient valve)*
- Safe mode on faults *(close all valves)*



```
// Decision 1: Should we irrigate?
let need_irrigation = safe_moisture < moisture_low_threshold in
let can_irrigate = s.tankB.level >= min_irrigation_amount in
let irrigation_not_maxed = s.irrigation_counter < max_consecutive_valve_open in
let irrigation_valve =
  if need_irrigation && can_irrigate && irrigation_not_maxed
  then Open else Closed
```



```
// Step 3: Controller logic (all valves closed if safe mode)
if enter_safe_mode then
  { s with
    valveAB = Closed;
    valveCB = Closed;
    irrigationValve = Closed;
    valveSupplyA = Closed;
    valveSupplyC = Closed;
    sensor_status = sensor_st;
    safe_mode = true;
```

Safety Properties

- No tank overflow
- Moisture always bounded
- Faulty sensors trigger safe mode

Liveness & Conclusions

- Guaranteed progress under assumptions
- Formal guarantees for CPS
- Future extensions